Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'tgmath.h.0p' command

$ man tgmath.h.0p

tgmath.h(0P) POSIX Programmer's Manual tgmath.h(0P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

tgmath.h ? type-generic macros

SYNOPSIS

#include <tgmath.h>

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1?2017 defers to the ISO C standard.

The <tgmath.h> header shall include the headers <math.h> and <com? plex.h> and shall define several type-generic macros.

Of the functions contained within the <math.h> and <complex.h> headers without an f (float) or l (long double) suffix, several have one or more parameters whose corresponding real type is double. For each such function, except modf(), j0(), j1(), jn(), y0(), y1(), and yn(), there shall be a corresponding type-generic macro. The parameters whose cor? responding real type is double in the function synopsis are generic pa?

rameters. Use of the macro invokes a function whose corresponding real type and type domain are determined by the arguments for the generic parameters.

Use of the macro invokes a function whose generic parameters have the corresponding real type determined as follows:

* First, if any argument for generic parameters has type long double, the type determined is long double.
* Otherwise, if any argument for generic parameters has type double or is of integer type, the type determined is double.
* Otherwise, the type determined is float.

For each unsuffixed function in the <math.h> header for which there is a function in the <complex.h> header with the same name except for a c prefix, the corresponding type-generic macro (for both functions) has the same name as the function in the <math.h> header. The corresponding type-generic macro for fabs() and cabs() is fabs().

<math.h> Function	<complex.h> Function	Type-Generic Macro
acos()	cacos()	acos()
asin()	casin()	asin()
atan()	catan()	atan()
acosh()	cacosh()	acosh()
asinh()	casinh()	asinh()
atanh()	catanh()	atanh()
cos()	ccos()	cos()
sin()	csin()	sin()
tan()	ctan()	tan()
cosh()	ccosh()	cosh()
sinh()	csinh()	sinh()
tanh()	ctanh()	tanh()
exp()	cexp()	exp()
log()	clog()	log()
pow()	cpow()	pow()

?sqrt() ? csqrt() ? sqrt() ?

?fabs() ? cabs() ? fabs() ?

??

If at least one argument for a generic parameter is complex, then use of the macro invokes a complex function; otherwise, use of the macro invokes a real function.

For each unsuffixed function in the <math.h> header without a c-prefixed counterpart in the <complex.h> header, except for modf(), j0(), j1(), jn(), y0(), y1(), and yn(), the corresponding type-generic macro has the same name as the function. These type-generic macros are:

atan2() fma() llround() remainder()

cbrt() fmax() log10() remquo()

ceil() fmin() log1p() rint()

copysign() fmod() log2() round()

erf() frexp() logb() scalbln()

erfc() hypot() lrint() scalbn()

exp2() ilogb() lround() tgamma()

expm1() ldexp() nearbyint() trunc()

fdim() lgamma() nextafter()

floor() llrint() nexttoward()

If all arguments for generic parameters are real, then use of the macro invokes a real function; otherwise, use of the macro results in undefined behavior.

For each unsuffixed function in the <complex.h> header that is not a c-prefixed counterpart to a function in the <math.h> header, the corresponding type-generic macro has the same name as the function. These type-generic macros are:

carg() cimag() conj() cproj() creal()

Use of the macro with any real or complex argument invokes a complex function.

The following sections are informative.

APPLICATION USAGE

With the declarations:

```
#include <tgmath.h>
int n;
float f;
double d;
long double ld;
float complex fc;
double complex dc;
long double complex ldc;
```

functions invoked by use of type-generic macros are shown in the fol? lowing table:

Macro	Use Invokes
?exp(n)	? exp(n), the function ?
?acosh(f)	? acoshf(f) ?
?sin(d)	? sin(d), the function ?
?atan(ld)	? atanl(ld) ?
?log(fc)	? clogf(fc) ?
?sqrt(dc)	? csqrt(dc) ?
?pow(ldc,f)	? cpowl(ldc, f) ?
?remainder(n,n)	? remainder(n, n), the function ?
?nextafter(d,f)	? nextafter(d, f), the function ?
?nexttoward(f,ld)	? nexttowardf(f, ld) ?
?copysign(n,ld)	? copysignl(n, ld) ?
?ceil(fc)	? Undefined behavior ?
?rint(dc)	? Undefined behavior ?
?fmax(ldc,ld)	? Undefined behavior ?
?carg(n)	? carg(n), the function ?
?cproj(f)	? cprojf(f) ?
?creal(d)	? creal(d), the function ?
?cimag(ld)	? cimagl(ld) ?
?cabs(fc)	? cabsf(fc) ?
?carg(dc)	? carg(dc), the function ?

?cproj(ldc) ? cprojl(ldc) ?

??

RATIONALE

Type-generic macros allow calling a function whose type is determined by the argument type, as is the case for C operators such as '+' and '*'. For example, with a type-generic cos() macro, the expression cos((float)x) will have type float. This feature enables writing more portably efficient code and alleviates need for awkward casting and suffixing in the process of porting or adjusting precision. Generic math functions are a widely appreciated feature of Fortran.

The only arguments that affect the type resolution are the arguments corresponding to the parameters that have type double in the synopsis. Hence the type of a type-generic call to nexttoward(), whose second pa? rameter is long double in the synopsis, is determined solely by the type of the first argument.

The term ``type-generic'' was chosen over the proposed alternatives of intrinsic and overloading. The term is more specific than intrinsic, which already is widely used with a more general meaning, and reflects a closer match to Fortran's generic functions than to C++ overloading.

The macros are placed in their own header in order not to silently break old programs that include the <math.h> header; for example, with:

```
printf ("%e", sin(x))
```

modf(double, double *) is excluded because no way was seen to make it safe without complicating the type resolution.

The implementation might, as an extension, endow appropriate ones of the macros that POSIX.1?2008 specifies only for real arguments with the ability to invoke the complex functions.

POSIX.1?2008 does not prescribe any particular implementation mechanism for generic macros. It could be implemented simply with built-in macros. The generic macro for sqrt(), for example, could be implemented with:

```
#undef sqrt
#define sqrt(x) __BUILTIN_GENERIC_sqrt(x)
```

Generic macros are designed for a useful level of consistency with C++ overloaded math functions.

The great majority of existing C programs are expected to be unaffected when the <tgmath.h> header is included instead of the <math.h> or <com? plex.h> headers. Generic macros are similar to the ISO/IEC 9899:1999 standard library masking macros, though the semantic types of return values differ.

The ability to overload on integer as well as floating types would have been useful for some functions; for example, copysign(). Overloading with different numbers of arguments would have allowed reusing names; for example, remainder() for remquo(). However, these facilities would have complicated the specification; and their natural consistent use, such as for a floating abs() or a two-argument atan(), would have in? troduced further inconsistencies with the ISO/IEC 9899:1999 standard for insufficient benefit.

The ISO C standard in no way limits the implementation's options for efficiency, including inlining library functions.

FUTURE DIRECTIONS

None.

SEE ALSO

<math.h>, <complex.h>

The System Interfaces volume of POSIX.1?2017, cabs(), fabs(), modf()

COPYRIGHT

most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.ker?nel.org/doc/man-pages/reporting_bugs.html .